Supervised Compression for Resource-Constrained Edge Computing Systems

Yoshitomo Matsubara   Ruihan Yang   Marco Levorato   Stephan Mandt
Department of Computer Science, University of California, Irvine
{yoshitom, ruihan.yang, levorato, mandt}@uci.edu

Abstract

There has been much interest in deploying deep learning algorithms on low-powered devices, including smartphones, drones, and medical sensors. However, full-scale deep neural networks are often too resource-intensive in terms of energy and storage. As a result, the bulk part of the machine learning operation is therefore often carried out on an edge server, where the data is compressed and transmitted. However, compressing data (such as images) leads to transmitting information irrelevant to the supervised task. Another popular approach is to split the deep network between the device and the server while compressing intermediate features. To date, however, such split computing strategies have barely outperformed the aforementioned naive data compression baselines due to their inefficient approaches to feature compression. This paper adopts ideas from knowledge distillation and neural image compression to compress intermediate feature representations more efficiently. Our supervised compression approach uses a teacher model and a student model with a stochastic bottleneck and learnable prior for entropy coding (Entropic Student). We compare our approach to various neural image and feature compression baselines in three vision tasks and found that it achieves better supervised rate-distortion performance while maintaining smaller end-to-end latency. We furthermore show that the learned feature representations can be tuned to serve multiple downstream tasks.

1. Introduction

With the abundance of smartphones, autonomous drones, and other intelligent devices, advanced computing systems for machine learning applications have become evermore important [47, 10]. Machine learning models are frequently deployed on low-powered devices for reasons of computational efficiency or data privacy [45, 25]. However, deploying conventional computer vision or NLP models on such hardware raises a computational challenge, as powerful deep neural networks are often too energy-consuming to be deployed on such weak mobile devices [17].

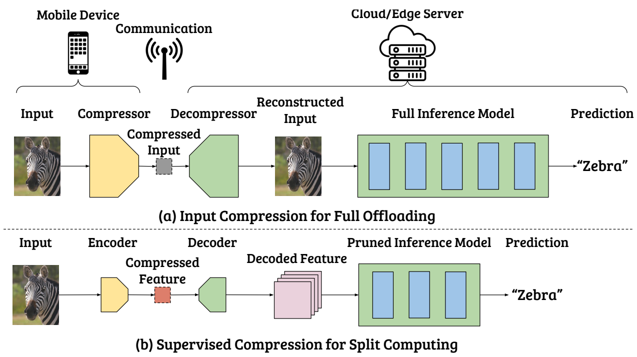


Figure 1: Image classification with input compression (**top**) vs. our proposed supervised compression for split computing (**bottom**). While the former approach fully reconstructs the image, our approach learns an intermediate compressible representation suitable for the supervised task.

An alternative to carrying out the deep learning model's operation on the low-powered device is to send compressed data to an edge server that takes care of the heavy computation instead. However, recent neural or classical compression algorithms are either resource-intensive [48, 16] and/or optimized for perceptual quality [4, 38], and therefore much of the information transmitted is redundant for the machine learning task [13] (see Fig. 1). A better solution is to therefore split the neural network [28] into the two sequences so that some elementary feature transformations are applied by the first sequence of the model on the weak mobile (local) device. Then, intermediate, informative features are transmitted through a wireless communication channel to the edge server that processes the bulk part of the computation (the second sequence of the model) [18, 35].

Traditional split computing approaches transmit intermediate features by either reducing channels in convolution layers [28] or truncating them to a lower arithmetic precision [36, 46, 37]. Since the models were not "informed" about such truncation steps during training oftentimes leads to substantial performance degradation. This raises the question of whether *learnable* end-to-end data compression pipelines can be designed to both truncate *and* entropy-code the involved early-stage features.

In this paper, we propose such a neural feature compression approach by drawing on variational inference-based data compression [5, 48]. Our architecture resembles the variational information bottleneck objective [1] and relies on an encoder, a "prior" on the bottleneck state, and a decoder that leads to a supervised loss (see Fig. 1). At inference time, we discretize the encoder's output and use our learned prior as a model for entropy coding intermediate features. The decoder reconstructs the feature vector losslessly from the binary bitstring and carries out the subsequent supervised machine learning task. Crucially, we combine this feature compression approach with knowledge distillation, where a teacher model provides the training data as well as parts of the trained architecture.[1]

In more detail, our main contributions are as follows:

- We propose a new training objective for feature compression in split computing that allows us to use a learned entropy model for bottleneck quantization in conjunction with knowledge distillation.
- Our approach significantly outperforms seven strong baselines from the split computing and (neural) image compression literature in terms of rate-distortion performance (with distortion measuring a supervised error) and in terms of end-to-end latency.
- Moreover, we show that a single encoder network can serve multiple supervised tasks, including classification, object detection, and semantic segmentation.

2. Related Work

Neural Image Compression. Neural image compression methods apply neural networks for nonlinear dimensionality reduction and subsequent entropy coding. Early works [50, 27] leveraged LSTMs to model spatial correlations of the pixels within an image. The first proposed autoencoder architecture for image compression [49] used the straight-through estimator [8] for learning a discrete latent representation. The connection of image compression to *probabilistic* generative models was drawn by variational autoencoders (VAEs) [29, 4]. In the subsequent work [5], two-level VAE architectures involving a scale hyper-prior are proposed to encode images, which can be further improved by autoregressive structures [38, 39] or by optimization at encoding time [52]. Recent work also shows the potential progressive compression of the VAE structure by extending the quantization grid [33]. Other works [53, 19] demonstrate competitive image compression performance without a pre-defined quantization grid.

Recently, Dubois *et al.* [16] propose a self-supervised compression architecture for generic image classification. However, their encoder involves 87.8 million parameters (627 times larger than our encoder in Table 1) due to its Vision Transformer (ViT [15])-based encoder used in CLIP model [41] for ImageNet dataset. Thus, it does not satisfy resource-constrained edge computing systems.

Split Computing. Given that mobile (or local) devices often have limited resources such as computing power and battery, we usually transfer sensor data captured by the mobile device and offload heavy computing tasks to an edge (or cloud) server with more computing resources. Unlike local computing, which executes the entire model on the mobile device, edge computing (or full offloading) where the computation is on the edge server requires quality wireless communication between the mobile device and edge server. Otherwise, the total inference cost, such as the end-to-end latency, would be higher than local computing due to the communication delay, which would be critical for real-time applications. As an intermediate option between local computing and edge computing, split computing [28] has been attracting attention from research communities since edge computing is not always the best option. Specifically, the communication cost would be a severe problem for resource-limited edge computing systems [18, 35].

In split computing, a neural model will be split into the first and second sequences, and the first sequence of the model is executed on the mobile device. Having received the output of the first section via wireless communication, the second sequence of the model completes the inference on the edge server. A key concept is to reduce computational load on the mobile device while saving communication cost (data size) as processing delay on the edge server is often smaller compared to local processing and communication delays [37]. For reducing communication cost, recent studies on split computing [18, 35, 46, 37] introduce *bottleneck*, whose data size is smaller than input sample, to vision models. In recent studies, a combination of 1) fewer channels (channel reduction) in convolution layers and 2) quantization at bottleneck point is key to design such bottlenecks. While such studies show the effectiveness of their proposed approach for image classification and object detection tasks, the accuracy of bottleneck-injected models sharply drops when further compressing the bottleneck size as we will describe in Section 4.

Knowledge Distillation. It is widely known that deep neural models are often overparameterized [2, 51], and knowledge distillation [24] is one of the well-known techniques for model compression [9]. In the paradigm, a large pretrained model plays a role of *teacher* for a model to be trained (called *student*), and the student model learns from both *hard*-targets (*e.g.*, one-hot vectors) and *soft*-targets (outputs of the teacher for the given input) during training.

[1] Code and models are available at https://github.com/yoshitomo-matsubara/supervised-compression

Interestingly, some uncertainty from the pretrained teacher model as *soft*-target is informative to student models. The models trained with teachers often achieve better prediction performance than those trained without teachers [2]. As will be discussed later in this paper, learning compressed features for a target task such as image classification [48] would be challenging, specifically in the case that we introduce such bottlenecks to early layers in the model. We leverage a pretrained model as the teacher, and those with introduced bottlenecks as students to be trained to improve rate-distortion performance.

3. Method

After providing an overview of the setup (Section 3.1) we describe our distillation and feature compression approach (Section 3.2) and our procedure to fine-tune the model to other supervised downstream tasks (Section 3.3).

3.1. Overview

Our goal is to learn a lightweight, communication-efficient feature extractor for supervised downstream applications. We thereby transmit intermediate feature activations between two distinct portions of a neural network. The first part is deployed on a low-power mobile device, and the second part on a compute-capable edge server. Intermediate feature representations are compressed and transmitted between the mobile device and the edge server using discretization and subsequent lossless entropy coding.

In order to learn good compressible feature representations, we combine two ideas: *knowledge distillation* and neural data compression via *learned entropy coding*. First, we train a large teacher network on a data set of interest to teach a smaller student model. We assume that the features that the teacher model learns are helpful for other downstream tasks. Then, we train a lightweight student model to match the teacher model's intermediate features (Section 3.2) with minimal performance loss. Finally, we fine-tune the student model to different downstream tasks (Section 3.3). Note that the training process is done offline.

The teacher network realizes a deterministic mapping $\mathbf{x} \mapsto \mathbf{h} \mapsto \mathbf{y}$, where $\mathbf{x}$ are the input data, $\mathbf{y}$ are the targets, and $\mathbf{h}$ are some intermediate feature representations of the teacher network. We assume that the teacher model is too large to be executed on the mobile device. The main idea is to replace the teacher model's mapping $\mathbf{x} \mapsto \mathbf{h}$ with a student model (*i.e.*, the new targets become the teacher model's intermediate feature activations). To facilitate the data transmission from the mobile device to the edge server, the student model, *Entropic Student*, embeds a bottleneck representation $\mathbf{z}$ that allows compression (see details below), and we transmit data as $\mathbf{x} \mapsto \mathbf{z} \mapsto \mathbf{h} \mapsto \mathbf{y}$. We show that the student model can be fine-tuned to different tasks while the mobile device's encoder part remains unchanged.

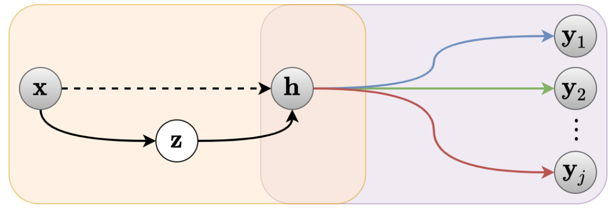


Figure 2: Proposed graphical model. Black arrows indicate the compression and decompression process in our student model. The dashed arrow shows the teacher's original deterministic mapping. Colored arrows show the discriminative tail portions shared between student and teacher.

The whole pipeline is visualized in the bottom panel of Fig. 1. The latent representation $\mathbf{z}$ has a "prior" $p(\mathbf{z})$, *i.e.*, a density model over the latent space $\mathbf{z}$ that both sender and receiver can use for entropy coding after discretizing $\mathbf{z}$. In the following, we derive the details of the approach.

3.2. Knowledge Distillation

We first focus on the details of the distillation process. The entropic student model learns the mapping $\mathbf{x} \mapsto \mathbf{h}$ (Fig. 2, left part) by drawing samples from the teacher model. The second part of the pipeline $\mathbf{h} \mapsto \mathbf{y}$ (Fig. 2, right part) will be adapted from the teacher model and will be fine-tuned to different tasks (see Section 3.3).

Similar to neural image compression [4, 5], we draw on latent variable models whose latent states allow us to quantize and entropy-code data under a prior probability model. In contrast to neural image compression, our approach is supervised. As such, it mathematically resembles the deep Variational Information Bottleneck [1] (which was designed for adversarial robustness rather than compression).

Distillation Objective. We assume a stochastic encoder $q(\mathbf{z}|\mathbf{x})$, a decoder $p(\mathbf{h}|\mathbf{z})$, and a density model ("prior") $p(\mathbf{z})$ in the latent space. Specific choices are detailed below. Similar to [1], we *maximize* mutual information between $\mathbf{z}$ and $\mathbf{h}$ (making the compressed bottleneck state $\mathbf{z}$ as informative as possible about the supervised target $\mathbf{h}$) while *minimizing* the mutual information between the input $\mathbf{x}$ and $\mathbf{z}$ (thus "compressing away" all the irrelevant information that does not immediately serve the supervised goal).

The objective for a given training pair $(\mathbf{x}, \mathbf{h})$ provided by the teacher model is

$$\mathcal{L}(\mathbf{x}, \mathbf{h}) = -\underbrace{\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log p_\phi(\mathbf{h}|\mathbf{z})}_{\text{distortion}} + \beta \underbrace{\log p_\phi(\mathbf{z})}_{\text{rate}}]. \quad (1)$$

Before discussing the rate and distortion terms, we specify and simplify this loss function further. Above, the decoder $p(\mathbf{h}|\mathbf{z}) = \mathcal{N}(\mathbf{h}; g_\phi(\mathbf{z}), I)$ is chosen as a conditional Gaussian centered around a deterministic prediction $g_\phi(\mathbf{z})$. Fol-

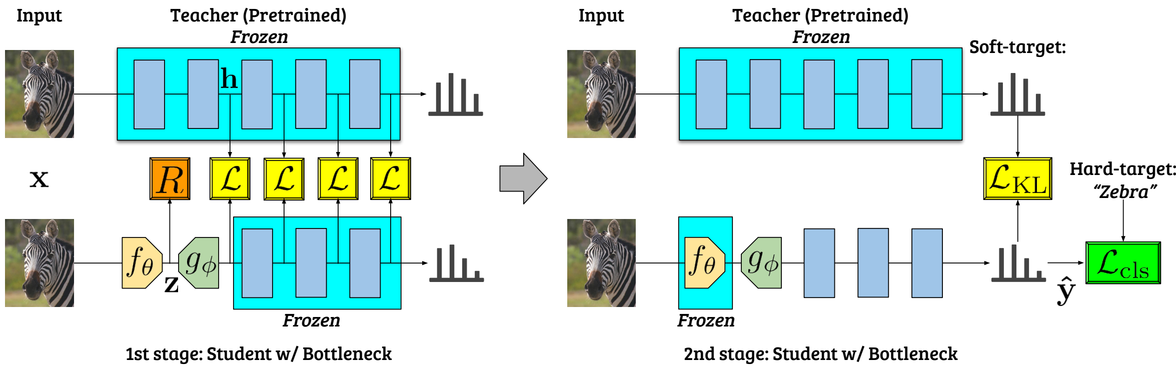


Figure 3: Our two-stage training approach. **Left**: training the student model (**bottom**) with targets $\mathbf{h}$ and tail architecture obtained from teacher (**top**) (Section 3.2). **Right**: fine-tuning the decoder and tail portion with fixed encoder (Section 3.3).

lowing the neural image compression literature [4, 5], the *encoder* is chosen to be a unit-width box function $q_\theta(\mathbf{z}|\mathbf{x}) = \mathcal{U}(f_\theta(\mathbf{x}) - \frac{1}{2}, f_\theta(\mathbf{x}) + \frac{1}{2})$ centered around a neural network prediction $f_\theta(\mathbf{x})$. Using the reparameterization trick [29], Eq. 1 can be optimized via stochastic gradient descent [2]

$$\mathcal{L}(\mathbf{x}, \mathbf{h}) = \frac{1}{2} \underbrace{||\mathbf{h} - g_\phi\left(f_\theta\left(\mathbf{x}\right) + \epsilon\right)||_2^2}_{\text{distortion}} \\ - \beta \underbrace{\log p_\phi(f_\theta(\mathbf{x}) + \epsilon)}_{\text{rate}}, \quad \epsilon \sim \text{Unif}(-\tfrac{1}{2}, \tfrac{1}{2}). \quad (2)$$

Once the model is trained, we discretize the latent state $\mathbf{z} = \lfloor f_\theta(\mathbf{x}) \rceil$ (where $\lfloor \cdot \rceil$ denotes the rounding operation) to allow for entropy coding under $p_\phi(\mathbf{z})$. By injecting noise from a box-shaped distribution of width one, we simulate the rounding operation during training. The entropy model or prior $p_\phi(\mathbf{z})$ is adopted from the neural image compression literature [5]; it is a learnable prior with tuning parameters ϕ. The prior factorizes over all dimensions of $\mathbf{z}$, allowing for efficient and parallel entropy coding.

Supervised Rate-Distortion Tradeoff. Similar to unsupervised data compression, our approach results in a rate-distortion tradeoff: the more aggressively we compress the latent representation $\mathbf{z}$, the more the predictive performance will deteriorate. In contrast, the more bits we are willing to invest for compressing $\mathbf{z}$, the more predictive strength our model will maintain. The goal will be to perform well on the unavailable tradeoff between rate and distortion.

The first term in Eq. 1 measures the supervised distortion, as it expresses the average prediction error under the coding procedure of first mapping $\mathbf{x}$ to $\mathbf{z}$ and then $\mathbf{z}$ to $\mathbf{h}$. In contrast, the second term measures the coding costs as the cross-entropy between the empirical distribution of $\mathbf{z}_i$ and the prior distribution $p_\phi(\mathbf{z})$ according to information theory [14]. The tradeoff is determined by the Lagrange multiplier β.

As a particular instantiation of an information bottleneck framework [1], Singh *et al.* [48] proposed a similar loss function as Eq. 1 to train a classifier with a bottleneck at its penultimate layer without knowledge distillation. In Section 4, we compare against a version of this approach that is compatible with our architecture and find that the knowledge distillation aspect is crucial to improve performance.

3.3. Fine-tuning for Target Tasks

Equation 1 shows the base approach, describing the knowledge distillation pipeline with a single $\mathbf{h}$ and involving a single target $\mathbf{y}$. In practice, our goal is to learn a compressed representation $\mathbf{z}$ that does not only serve a single supervised target $\mathbf{y}$, but multiple ones $\mathbf{y}_1, \cdots, \mathbf{y}_j$. In particular, for a deployed system with a learned compression module, we would like to be able to fine-tune the part of the network living on the edge server to multiple tasks without having to retrain the compression model. As follows, we show that such multi-task learning is possible.

A learned student model from knowledge distillation can be depicted as a two-step deterministic mapping $\mathbf{z} = \lfloor f_\theta(\mathbf{x}) \rceil$ and $\hat{\mathbf{h}} = g_\phi(\mathbf{z})$, where $\hat{\mathbf{h}}$ ($\approx \mathbf{h}$) is now a decompressed intermediate hidden feature in our final student model (see Fig. 2). Assuming that $p_{\psi_j}(\mathbf{y}_j|\hat{\mathbf{h}})$ denotes the student model's output probability distribution with parameters ψ_j, the fine-tuning step amounts to optimizing

$$\psi_j^* = \arg\min_{\psi_j} -\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}}[p_{\psi_j}(\mathbf{y}_j|g_\phi(\lfloor f_\theta(\mathbf{x}) \rceil))]. \quad (3)$$

The pair $(\mathbf{y}_j, \psi_j)$ refers to the target label and the parameters of each downstream task. The formula illustrates the Maximum Likelihood Estimation (MLE) method to optimize the parameter ψ_j for task j. Note that we optimize the discriminative model after the compression model is frozen, so θ is fixed in this training stage, and ϕ can either be fixed or trainable. We elucidate the hybrid model in Fig. 2.

[2]For better convergence, we follow [37] and leverage intermediate representations from frozen layers besides $\mathbf{h}$ as illustrated in Fig. 3 (left).

For fine-tuning the student model with the frozen encoder, we leverage a teacher model again. For image classification, we apply a standard knowledge distillation technique [24] to achieve better model accuracy by distilling the knowledge in the teacher model into our student model. Specifically, we fine-tune the student model by minimizing a weighted sum of two losses: 1) cross-entropy loss between the student model's class probability distribution and one-hot vector (*hard*-target), and 2) Kullback-Leibler divergence between *softened* class probability distributions from both the student and teacher models.

Similarly, having frozen the encoder, we can fine-tune different models for different downstream tasks reusing the trained entropic student model (classifier) as their backbone, which will be demonstrated in Section 4.4.

4. Experiments

Using torchdistill [34], we designed different experiments and studied various models based on both principles of split-computing (partial offloading) and edge computing (full offloading). We used ResNet-50 [23] as a base model, which, besides image classification, is also widely used as a backbone for different vision tasks such as object detection [22, 31] and semantic segmentation [12]. In all experiments, we empirically show that our approach leads to better supervised rate-distortion performance.

4.1. Baselines

In this study, we use seven baseline methods categorized into either input compression or feature compression.

Input compression (IC). A conventional implementation of the edge computing paradigm is to transmit the compressed image directly to the edge server, where all the tasks are then executed. We consider five baselines referring to this "input compression" scenario: JPEG, WebP [20], BPG [7], and two neural image compression methods (factorized prior and mean-scale hyperprior) [5, 38] based on CompressAI [6]. The latter approach is currently considered state of the art in image compression models (without autoregressive structure) [38, 39, 52]. We evaluate each model's performance in terms of the rate-distortion curve by setting different quality values for JPEG, WebP, and BPG and Lagrange multiplier β for neural image compression.

Feature compression (FC). Split computing baselines [36, 46] correspond to reducing the bottleneck data size with channel reduction and bottleneck quantization referred to as CR+BQ (quantizes 32-bit floating-point to 8-bit integer) [26]. Matsubara *et al.* [37, 36] report that bottleneck quantization did not lead to significant accuracy loss. To control the rate-distortion tradeoff, we design bottlenecks with a different number of output channels in a convolution layer to control the bottleneck data size, train the bottleneck-injected models and quantize the bottleneck after the training session.

Our final baseline in this paper is an end-to-end approach towards learning compressible features for a single task similar to Singh *et al.* [48] (for brevity, we will cite their reference). Their originally proposed approach focuses only on classification and introduces the compressible bottleneck to the penultimate layer. In the considered setting, such design leads to an overwhelming workload for the mobile/local device: for example, in terms of model parameters, about 92% of the ResNet-50 [23] parameters would be deployed on the weaker, mobile device. To make this approach compatible with our setting, we apply their approach to our architecture; that is, we directly train our entropic student model without a teacher model. We find that compared to [48], having a stochastic bottleneck at an earlier layer (due to limited capacity of mobile devices) leads to a model that is much harder to optimize (see Section 4.3).

4.2. Implementation of Our Entropic Student

Vision models in recent years reuse pretrained image classification models as their backbones *e.g.*, ResNet-50 [23] as a backbone of RetinaNet [31] and Faster R-CNN [42] for object detection tasks. These models often use intermediate hidden features extracted from multiple layers in the backbone as the input to subsequent task-specific modules such as feature pyramid network (FPN) [30]. Thus, using an architecture with a bottleneck introduced at late layers [48] for tasks other than image classification may require transferring and compressing multiple hidden features to an edge server, which will result in high communication costs.

To improve the efficiency of split computing compared to that of edge computing, we introduce the bottleneck as early in the model as possible to reduce the computational workload at the mobile device. We replace the first layers of our pretrained teacher model with the new modules for encoding and decoding transforms as illustrated in Fig. 3. The student model, *entropic student*, consists of the new modules and the remaining layers copied from its teacher model for initialization. Similar to neural image compression models [5, 38], we use convolution layers and simplified generalized divisive normalization (GDN) [3] layers to design an encoder f_θ, and design a decoder g_ϕ with convolution and inverse version of simplified GDN (IGDN) layers. Importantly, the designed encoder should be lightweight, *e.g.*, with fewer model parameters as it will be deployed and executed on a low-powered mobile device. We will discuss the deployment cost in Section 4.6.

Different from bottleneck designs in the prior studies on split computing [37, 36], we control the trade-off between bottleneck data size and model accuracy with the β value in the rate-distortion loss function (See Eq. 2).

4.3. Image Classification

We first discuss the rate-distortion performance of our and baseline models using a large-scale image classification dataset. Specifically, we use ImageNet (ILSVRC 2012) [43], that consists of 1.28 million training and 50,000 validation samples. As is standard, we train the models on the training split and report the top-1 accuracy on the validation split. Using ResNet-50 [23] pre-trained on ImageNet as a teacher model, we replace all the layers before its second residual block with our encoder and decoder to compose our *entropic student* model. The introduced encoder-decoder modules are trained to approximate $\mathbf{h}$ in Eq. 2, which is the output of the corresponding residual block in the teacher model (original ResNet-50) in the first stage, and then we fine-tune the student model as described in Section 3. We provide more details of training configurations (*e.g.*, hyperparameters) in the supplementary material.

Figure 4 presents supervised rate-distortion curves of ResNet-50 with various compression approaches, where the x-axis shows the average data size, and the y-axis the supervised performance. We note that the input tensor shape for ResNet-50 as an image classifier is $3 \times 224 \times 224$. For image compression, the result shows the considered neural compression models, factorized prior [5] and mean-scale hyperprior [38], consistently outperform JPEG and WebP compression in terms of rate-distortion curves in the image classification task. A popular approach used in split computing studies, the combination of channel reduction and bottleneck quantization (CR+BQ) [37], seems slightly better than JPEG compression but not as accurate as those with the neural image compression models.

Among all the configurations in the figure, our model trained by the two-stage method performs the best. We also trained our model without teacher model, which in essence corresponds to [48]. The resulting RD curve is significantly worse, which we attribute to two possible effects: first, it is widely acknowledged that knowledge distillation generally finds solutions that generalize better. Second, having a stochastic bottleneck at an earlier layer may make it difficult for the end-to-end training approach to optimize.

4.4. Object Detection and Semantic Segmentation

As suggested by He *et al.* [21], image classifiers pre-trained on the ImageNet dataset [43] speed up the convergence of training on downstream tasks. Reusing the proposed model pre-trained on the ImageNet dataset, we further discuss the rate-distortion performance on two downstream tasks: object detection and semantic segmentation. Specifically, we train RetinaNet [31] and DeepLabv3 [12], using our models pre-trained on the ImageNet dataset in the previous section as their backbone. RetinaNet is a one-stage object detection model that enables faster inference than two-stage detectors such as Mask R-CNN [22]. DeepLabv3 is a semantic segmentation model that leverages Atrous Spatial Pyramid Pooling (ASPP) [11].

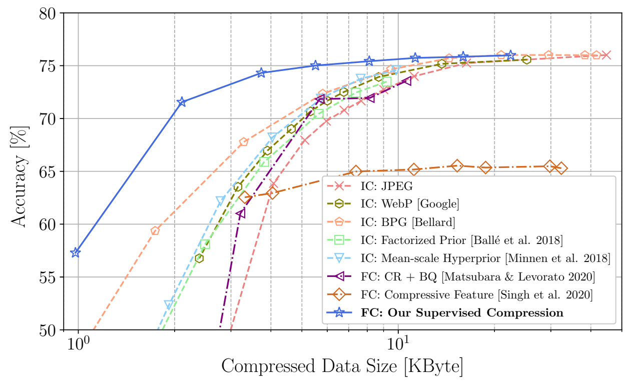


Figure 4: Rate-distortion (accuracy) curves of ResNet-50 as base model for ImageNet (ILSVRC 2012).

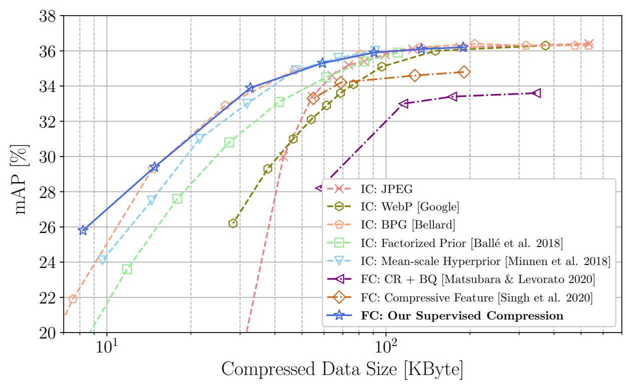


Figure 5: Rate-distortion (BBox mAP) curves of RetinaNet with ResNet-50 and FPN as base backbone for COCO 2017.

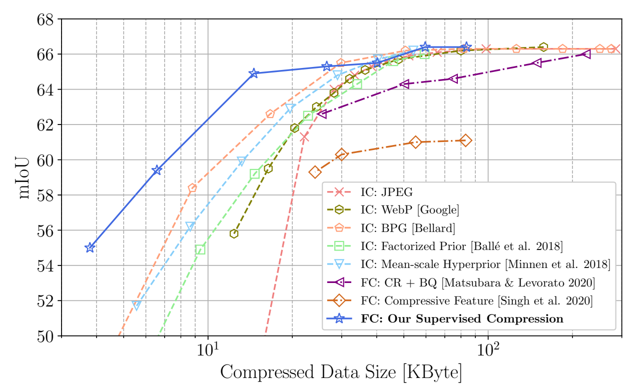


Figure 6: Rate-distortion (Seg mIoU) curves of DeepLabv3 with ResNet-50 as base backbone for COCO 2017.

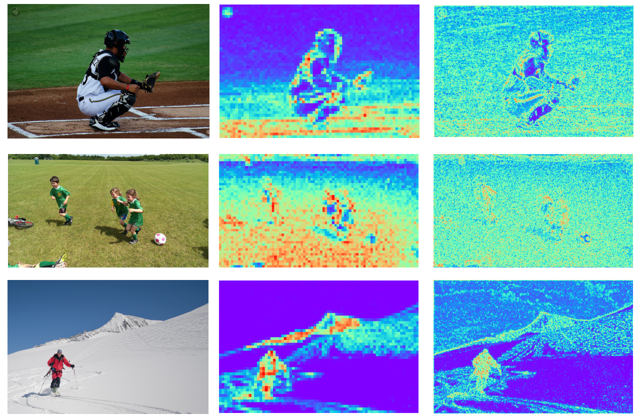
For the downstream tasks, we use the COCO 2017 dataset [32] to fine-tune the models. The training and validation splits in the COCO 2017 dataset have 118,287 and 5,000 annotated images, respectively. As detection performance, we refer to mean average precision (mAP) for bounding box (BBox) outputs with different Intersection-over-Unions (IoU) thresholds from 0.5 and 0.95 on the validation split. For semantic segmentation, we measure the performance by pixel IoU averaged over 21 classes present in the PASCAL VOC 2012 dataset. It is worth noting that following the PyTorch [40] implementations, the input image scales for RetinaNet [31] are defined by the shorter image side and set to 800 in this study which is much larger than the input image in the previous image classification task. As for DeepLabv3 [12], we use the resized input images such that their shorter size is 520. The training setup and hyperparameters used to fine-tune the models are described in the supplementary material.

Similar to the previous experiment for the image classification task, Figures 5 and 6 show that the combinations of neural compression models and the pre-trained RetinaNet and DeepLabv3, which are still strong baselines in object detection and semantic segmentation tasks. Our model demonstrates better rate-distortion curves in both tasks. In the object detection task, our model's improvements over RetinaNet with BPG and mean-scale hyperprior are smaller than those in the image classification and semantic segmentation tasks. However, our model's encoder to be executed on a mobile device is approximately 40 times smaller than the encoder of the mean-scale hyperprior. Our model also can achieve a much shorter latency to complete the input-to-prediction pipeline (see Fig. 1) than the baselines we considered for resource-constrained edge computing systems. We further discuss these aspects in Sections 4.6 and 4.7.

4.5. Bitrate Allocation of Latent Representations

This section discusses the difference between the representations of bottlenecks in neural image compression and our models. We are interested in which element of the bottlenecks allocates more bits in the latent representation $\mathbf{z}$. Bottlenecks in neural image compression models will allocate many bits to some *unique* area in an image to preserve all its characteristics in the reconstructed image. On the other hand, those in our models are trained to mimic the feature representations in their teacher model, thus expected to allocate more bits to areas useful for the target task.

Figure 7 shows visualizations of the normalized bitrate allocations for a few sample images. The 2nd column of the figure corresponds to the bottleneck in a neural image compression model prioritizing the images' backgrounds such as catcher's zone and glasses. Interestingly, our bottleneck representation (the 3rd column) seems to eliminate the difference between the two backgrounds and focuses on objects in the images such as persons and soccer ball. Moreover, the bottleneck eliminates a digital watermark at the top left in the first image, which is most likely not critical for the target task, while the one in the neural compression model noticeably distinguishes the logo from the background.



(a) Input Images (b) Factorized Prior (c) Entropic Student

Figure 7: Bitrate allocations of latent representations $\mathbf{z}$ in neural image compression and our entropic student models. Red and blue areas are allocated higher and lower bitrates, respectively (best viewed in PDF). It appears that the supervised approach (right) allocates more bits to the information relevant to the supervised classification goal.

Table 1: Number of parameters in compression and classification models loaded on mobile device and edge server. Local (LC), Edge (EC), and Split computing (SC).

Compression model	**Scenario**	**Model size (# params)**	
Factorized Prior [5]	EC	Mobile: 1.30M	Edge: 1.30M+*
Mean-Scale Hyperprior [38]	EC	Mobile: 5.53M	Edge: 4.49M+*
Classification model	**Scenario**	**Model size (# params)**	
MobileNetV2 [45]	LC	3.50M	
MobileNetV3 [25]	LC	5.48M	
ResNet-50 [23]	EC	25.6M	
ResNet-50 w/ BQ [37]	SC	Mobile: 0.01M	Edge: 27.3M
Our Entropic Student	SC	Mobile: 0.14M	Edge: 26.5M

* Size of classification model for EC should be additionally considered.

4.6. Deployment Cost on Mobile Devices

In Sections 4.3 and 4.4, we discussed the trade-off between transferred data size and model accuracy with the visualization of rate-distortion curves. While improving the rate-distortion curves is essential for resource-constrained edge computing systems, it is also important to reduce the computational burden allocated to the mobile device that often have more severe constraints on computing and energy resources compared to edge servers. We investigate then the cost of deploying image classification models on constrained mobile devices.

Table 1 summarizes numbers of parameters used to represent models deployed on the mobile devices and edge servers under different scenarios. For example, in the edge computing (EC) scenario, the input data compressed by the compressor of an input compression model is sent to the edge server. The decompressor will reconstruct the input data to complete the inference task with a full classification model. Thus, only the compressor in the input compression model is accounted for in the computation cost on the mobile device. In Section 4, the two input compression models, factorized prior [5] and mean-scale hyperprior [38], are strong baseline approaches, and mean-scale hyperprior outperforms the factorized prior in terms of rate-distortion curve. However, its model size is comparable to or more expensive than popular lightweight models such as MobileNetV2 [45] and MobileNetV3 [25]. For this reason, this strategy is not advantageous unless the model deployed on the edge server can offer much higher accuracy than the lightweight models on the mobile device.

In contrast, split computing (SC) models, including our entropic student model, perform in-network feature compression while extracting features from the target task's input sample. As shown in Table 1, the encoder of our model is much smaller (about 10 – 40 times smaller than) compared to those of the input compression models and the lightweight classifiers. Moreover, the encoder in our student model can be shared with RetinaNet and DeepLabv3 for different tasks. When a mobile device has multiple tasks such as image classification, object detection, and semantic segmentation, the single encoder is on memory and executed for an input sample. We note that ResNet-50 models with channel reduction and bottleneck quantization [37] and those for compressive feature [48] in their studies require a non-shareable encoder for different tasks. With their approaches, there are three individual encoders on the memory of the more constrained mobile device, which leads to approximately 3 times larger deployment cost.

4.7. End-to-End Prediction Latency Evaluation

To compare the prediction latency with the different approaches, we deploy the encoders on two different mobile devices: Raspberry Pi 4 (RPI4) and NVIDIA Jetson TX2 (JTX2). As an edge server (ES), we use a desktop computer with an NVIDIA GeForce RTX 2080 Ti, assuming the use of LoRa [44] for low-power communications (maximum data rate is 37.5 Kbps). For all the considered approaches, we use the data points with about 74% accuracy in Fig. 4, and the end-to-end latency is the sum of 1) execution time to encode an input image on RPI4/JTX2, 2) delay to transfer the encoded data from RPI4/JTX2 to ES, and 3) execution time to decode the compressed data and complete inference on ES.

Table 2 shows that our approach reduces the end-to-end prediction latency by 47 – 62% compared to the baselines. The encoding time and communication delay are dominant in the end-to-end latency while the execution time on ES is negligible. For both the experimental configurations (RPI4 $\longrightarrow$ ES and JTX2 $\longrightarrow$ ES), the breakdowns of the end-to-end latency are illustrated in the supplementary material.

Table 2: End-to-end latency to complete input-to-prediction pipeline for resource-constrained edge computing systems illustrated in Fig. 1, using RPI4/JTX2, LoRa and ES. The breakdowns are available in the supplementary material.

Approach	RPI4 $\longrightarrow$ ES	JTX2 $\longrightarrow$ ES
JPEG + ResNet-50	2.35 sec	2.34 sec
WebP + ResNet-50	1.83 sec	1.84 sec
BPG + ResNet-50	2.46 sec	2.41 sec
Factorized Prior + ResNet-50	2.43 sec	2.22 sec
Mean-Scale Hyperprior + ResNet-50	2.24 sec	1.92 sec
ResNet-50 w/ BQ	2.27 sec	2.25 sec
Our Entropic Student	**0.972 sec**	**0.904 sec**

5. Conclusions

This paper adopts ideas from knowledge distillation and neural image compression to achieve feature compression for supervised tasks. Our approach leverages a teacher model to introduce a stochastic bottleneck and a learnable prior for entropy coding at its early stage of a student model (namely, *Entropic Student*). The framework reduces the computational burden on the weak mobile device by offloading most of the computation to a computationally powerful cloud/edge server, and the single encoder in our entropic student can serve multiple downstream tasks. The experimental results show the improved supervised rate-distortion performance for three different vision tasks and the shortened end-to-end prediction latency, compared to various (neural) image compression and feature compression baselines.

To ensure reproducibility of the experimental results and facilitate research to address this important problem, we release the training code and trained models at `https://github.com/yoshitomo-matsubara/supervised-compression`.

Acknowledgements

This material is in part based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0021. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. We acknowledge the support by the Google Cloud Platform research credits, and the National Science Foundation under the NSF CAREER award 2047418 and Grants 1928718, 2003237, 2007719, IIS-1724331 and MLWiNS-2003237, as well as Intel and Qualcomm. We also thank Yibo Yang and the anonymous reviewers for their insightful comments.

References

[1] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep Variational Information Bottleneck. In *International Conference on Learning Representations*, 2017.

[2] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS 2014*, pages 2654–2662, 2014.

[3] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. Density Modeling of Images using a Generalized Normalization Transformation. In *International Conference on Learning Representations*, 2016.

[4] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end Optimized Image Compression. *International Conference on Learning Representations*, 2017.

[5] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *International Conference on Learning Representations*, 2018.

[6] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja. CompressAI: a PyTorch library and evaluation platform for end-to-end compression research. *arXiv preprint arXiv:2011.03029*, 2020. `https://github.com/InterDigitalInc/CompressAI`.

[7] Fabrice Bellard. BPG Image format. `https://bellard.org/bpg/` [Accessed on August 6, 2021].

[8] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv preprint arXiv:1308.3432*, 2013.

[9] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541, 2006.

[10] Jiasi Chen and Xukan Ran. Deep Learning With Edge Computing: A Review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[11] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):834–848, 2017.

[12] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[13] Jinyoung Choi and Bohyung Han. Task-aware quantization network for jpeg image compression. In *European Conference on Computer Vision*, pages 309–324. Springer, 2020.

[14] Thomas M Cover. *Elements of Information Theory*. John Wiley & Sons, 1999.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2020.

[16] Yann Dubois, Benjamin Bloem-Reddy, Karen Ullrich, and Chris J Maddison. Lossy Compression for Lossless Prediction. In *Neural Compression: From Information Theory to Applications–Workshop@ ICLR 2021*, 2021.

[17] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing*, 2019.

[18] Amir Erfan Eshratifar, Amirhossein Esmaili, and Massoud Pedram. BottleNet: A Deep Learning Architecture for Intelligent Mobile Cloud Computing Services. In *2019 IEEE/ACM Int. Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2019.

[19] Gergely Flamich, Marton Havasi, and José Miguel Hernández-Lobato. Compression without Quantization. In *OpenReview*, 2019.

[20] Google. Compression Techniques — WebP — Google Developers. `https://developers.google.com/speed/webp/docs/compression` [Accessed on August 6, 2021].

[21] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking ImageNet Pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019.

[22] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *Deep Learning and Representation Learning Workshop: NIPS 2014*, 2014.

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[26] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[27] Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen, Sung Jin Hwang, Joel Shor, and George Toderici. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4385–4393, 2018.

[28] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second Inter-*

national Conference on Architectural Support for Programming Languages and Operating Systems, pages 615–629, 2017.

[29] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2014.

[30] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[31] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[33] Yadong Lu, Yinhao Zhu, Yang Yang, Amir Said, and Taco S Cohen. Progressive Neural Image Compression with Nested Quantization and Latent Ordering. *arXiv preprint arXiv:2102.02913*, 2021.

[34] Yoshitomo Matsubara. torchdistill: A Modular, Configuration-Driven Framework for Knowledge Distillation. In *International Workshop on Reproducible Research in Pattern Recognition*, pages 24–44. Springer, 2021. `https://github.com/yoshitomo-matsubara/torchdistill`.

[35] Yoshitomo Matsubara, Sabur Baidya, Davide Callegaro, Marco Levorato, and Sameer Singh. Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems. In *Proc. of the 2019 MobiCom Workshop on Hot Topics in Video Analytics and Intelligent Edges*, pages 21–26, 2019.

[36] Yoshitomo Matsubara, Davide Callegaro, Sabur Baidya, Marco Levorato, and Sameer Singh. Head Network Distillation: Splitting Distilled Deep Neural Networks for Resource-Constrained Edge Computing Systems. *IEEE Access*, 8:212177–212193, 2020.

[37] Yoshitomo Matsubara and Marco Levorato. Neural Compression and Filtering for Edge-assisted Real-time Object Detection in Challenged Networks. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 2272–2279, 2021.

[38] David Minnen, Johannes Ballé, and George D Toderici. Joint Autoregressive and Hierarchical Priors for Learned Image Compression. In *Advances in Neural Information Processing Systems*, pages 10771–10780, 2018.

[39] David Minnen and Saurabh Singh. Channel-Wise Autoregressive Entropy Models for Learned Image Compression. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3339–3343. IEEE, 2020.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[41] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning Transferable Visual Models From Natural Language Supervision. *arXiv preprint arXiv:2103.00020*, 2021.

[42] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[43] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[44] Farzad Samie, Lars Bauer, and Jörg Henkel. IoT Technologies for Embedded Computing: A Survey. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–10. IEEE, 2016.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[46] Jiawei Shao and Jun Zhang. BottleNet++: An end-to-end approach for feature compression in device-edge co-inference systems. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE, 2020.

[47] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[48] Saurabh Singh, Sami Abu-El-Haija, Nick Johnston, Johannes Ballé, Abhinav Shrivastava, and George Toderici. End-to-end Learning of Compressible Features. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 3349–3353. IEEE, 2020.

[49] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy Image Compression with Compressive Autoencoders. In *International Conference on Learning Representations*, 2017.

[50] George Toderici, Damien Vincent, Nick Johnston, Sung Jin Hwang, David Minnen, Joel Shor, and Michele Covell. Full Resolution Image Compression with Recurrent Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5306–5314, 2017.

[51] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? In *Fifth International Conference on Learning Representations*, 2017.

[52] Yibo Yang, Robert Bamler, and Stephan Mandt. Improving Inference for Neural Image Compression. In *Advances in*

Neural Information Processing Systems, volume 33, pages 573–584, 2020.

[53] Yibo Yang, Robert Bamler, and Stephan Mandt. Variational Bayesian Quantization. In *International Conference on Machine Learning*, pages 10670–10680. PMLR, 2020.